

AD-A199 579

AVF Control Number: AVF-VSR-017  
SZ1-AVF-017

DTIC FILE COPY

Ada COMPILER  
VALIDATION SUMMARY REPORT:  
Certificate Number: 88030511.09046  
Siemens AG, Muenchen  
Siemens BS2000 Ada Compiler, V1.0  
Siemens 7.570P

Completion of On-Site Testing:  
5th March 1988

Prepared By:  
IABG m.b.H., Dept SZ1  
Einsteinstrasse 20  
8012 Ottobrunn  
West Germany

Prepared For:  
Ada Joint Program Office  
United States Department of Defense  
Washington, D.C. 20301-3081

DTIC  
ELECTE  
AUG 3 1988  
S H D

-----  
Ada is a registered trademark of the United States Government  
(Ada Joint Program Office).

DISTRIBUTION STATEMENT A  
Approved for public release;  
Distribution Unlimited

88 8 31 029

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: Siemens AG, Muenchen, Siemens BS2000 Ada Compiler, V1.0, Siemens 7.570P under BS2000, V7.6 (Host and Target).		5. TYPE OF REPORT & PERIOD COVERED 5 March 1988 to 5 March 1989
7. AUTHOR(s) IABG, Ottobrunn, Federal Republic of Germany		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION AND ADDRESS IABG, Ottobrunn, Federal Republic of Germany		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) IABG, Ottobrunn, Federal Republic of Germany		12. REPORT DATE 5 March 1988
		13. NUMBER OF PAGES 41 p.
		15. SECURITY CLASS (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT (of this Report)  Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20. If different from Report)  UNCLASSIFIED		
18. SUPPLEMENTARY NOTES		
19. KEYWORDS (Continue on reverse side if necessary and identify by block number)  Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD- 1815A, Ada Joint Program Office, AJPO		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)  Siemens BS2000 Ada Compiler, V1.0, Siemens AG, Muenchen, IABG, Siemens 7.570P under BS2000, V7.6 (Host and Target), ACVC 1.9.		

DD FORM

1473

EDITION OF 1 NOV 65 IS OBSOLETE

1 JAN 77

S/N 0102-LE-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Ada Compiler Validation Summary Report:

Compiler Name: Siemens BS2000 Ada Compiler, V1.0

Certificate Number: 88030511.09046

Host:

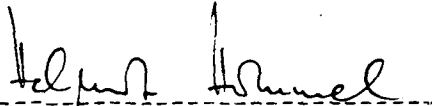
Siemens 7.570F under  
BS2000,  
V7.6

Target:

Siemens 7.570P under  
BS2000,  
V7.6

Testing Completed 5th March 1988 Using ACVC 1.9

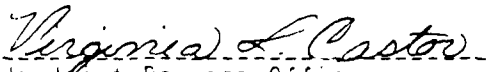
This report has been reviewed and is approved.



IABG m.b./H., Dept SZT  
Dr. H. Hummel  
Einsteinstrasse 20  
8012 Ottobrunn  
West Germany



Ada Validation Organization  
Dr. John F. Kramer  
Institute for Defense Analyses  
Alexandria VA 22311



Ada Joint Program Office  
Virginia L. Castor  
Director  
Department of Defense  
Washington DC 20301

Ada is a registered trademark of the United States Government  
(Ada Joint Program Office).

# CONTENTS

CHAPTER 1	INTRODUCTION	
1.1	PURPOSE OF THIS VALIDATION SUMMARY REPORT . . . . .	1-2
1.2	USE OF THIS VALIDATION SUMMARY REPORT . . . . .	1-2
1.3	REFERENCES . . . . .	1-3
1.4	DEFINITION OF TERMS . . . . .	1-4
1.5	ACVC TEST CLASSES . . . . .	1-5
CHAPTER 2	CONFIGURATION INFORMATION	
2.1	CONFIGURATION TESTED . . . . .	2-1
2.2	IMPLEMENTATION CHARACTERISTICS . . . . .	2-2
CHAPTER 3	TEST INFORMATION	
3.1	TEST RESULTS . . . . .	3-1
3.2	SUMMARY OF TEST RESULTS BY CLASS . . . . .	3-1
3.3	SUMMARY OF TEST RESULTS BY CHAPTER . . . . .	3-2
3.4	WITHDRAWN TESTS . . . . .	3-2
3.5	INAPPLICABLE TESTS . . . . .	3-2
3.6	TEST, PROCESSING, AND EVALUATION MODIFICATIONS . . . . .	3-4
3.7	ADDITIONAL TESTING INFORMATION . . . . .	3-5
3.7.1	Prevalidation . . . . .	3-5
3.7.2	Test Method . . . . .	3-5
3.7.3	Test Site . . . . .	3-6
APPENDIX A	DECLARATION OF CONFORMANCE	
APPENDIX B	APPENDIX F OF THE Ada STANDARD	
APPENDIX C	TEST PARAMETERS	
APPENDIX D	WITHDRAWN TESTS	



Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

## CHAPTER 1

### INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

## INTRODUCTION

### 1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- . To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- . To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- . To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 5th March 1988 at SIEMENS AG at Muenchen 83, Germany.

### 1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse  
Ada Joint Program Office  
OUSDRE  
The Pentagon, Rm 3D-139 (Fern Street)  
Washington DC 20301-3081

or from:

IABG m.b.H., Dept SZT  
Einsteinstrasse 20  
8012 Ottobrunn  
West Germany

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization  
Institute for Defense Analyses  
1801 North Beauregard Street  
Alexandria VA 22311

### 1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.
3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.
4. Ada Compiler Validation Capability User's Guide, December 1986.

## INTRODUCTION

### 1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada Commentary	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the <u>Ada Compiler Validation Procedures and Guidelines</u> .
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect



because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

## 1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce compilation or link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

## INTRODUCTION

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK\_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK\_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of REPORT and CHECK\_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

## CHAPTER 2

### CONFIGURATION INFORMATION

#### 2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: Siemens BS2000 Ada Compiler, V1.0

ACVC Version: 1.7

Certificate Number: B80305i1.09046

Host Computer:

Machine: Siemens 7.570P

Operating System: BS2000  
V7.6

Memory Size: 32 MB

Target Computer:

Machine: Siemens 7.570P

Operating System: BS2000  
V7.6

Memory Size: 32 MB

## CONFIGURATION INFORMATION

### 2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

- Capacities.

The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

An implementation is allowed to reject universal integer calculations having values that exceed `SYSTEM.MAX_INT`. This implementation processes 64 bit integer calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

This implementation supports the additional predefined type `SHORT_INTEGER`, in the package `STANDARD`. (See tests B86001C and B86001D.)

- Based literals.

An implementation is allowed to reject a based literal with a value exceeding `SYSTEM.MAX_INT` during compilation, or it may raise `NUMERIC_ERROR` or `CONSTRAINT_ERROR` during execution. This implementation raises `CONSTRAINT_ERROR` during execution. (See test E24101A.)

- Expression evaluation.

Apparently no default initialization expressions for record components are evaluated before any value is checked to belong to a component's subtype. (See test C32117A.)

Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

## CONFIGURATION INFORMATION

This implementation uses no extra bits for extra precision. This implementation uses all extra bits for extra range. (See test C35903A.)

Sometimes CONSTRAINT\_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

Apparently CONSTRAINT\_ERROR is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

Apparently underflow is not gradual. (See tests C45524A..Z.)

### Rounding.

The method used for rounding to integer is apparently round away from zero. (See tests C46012A..Z.)

The method used for rounding to longest integer is apparently round away from zero. (See tests C46012A..Z.)

The method used for rounding to integer in static universal real expressions is apparently round away from zero. (See test C4A014A.)

## CONFIGURATION INFORMATION

### Array types.

An implementation is allowed to raise `NUMERIC_ERROR` or `CONSTRAINT_ERROR` for an array having a `'LENGTH` that exceeds `STANDARD.INTEGER'LAST` and/or `SYSTEM.MAX_INT`. For this implementation:

Declaration of an array type or subtype declaration with more than `SYSTEM.MAX_INT` components raises no exception. (See test C36003A.)

`NUMERIC_ERROR` is raised when `'LENGTH` is applied to an array type with `INTEGER'LAST + 2` components. (See test C36202A.)

`NUMERIC_ERROR` is raised when `'LENGTH` is applied to an array type with `SYSTEM.MAX_INT + 2` components. (See test C36202B.)

A packed `BOOLEAN` array having a `'LENGTH` exceeding `INTEGER'LAST` raises `NUMERIC_ERROR` when the array type is declared. (See test C52103X.)

A packed two-dimensional `BOOLEAN` array with more than `INTEGER'LAST` components raises `NUMERIC_ERROR` when the array type is declared. (See test C52104Y.)

A null array with one dimension of length greater than `INTEGER'LAST` may raise `NUMERIC_ERROR` or `CONSTRAINT_ERROR` either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises `NUMERIC_ERROR` when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before `CONSTRAINT_ERROR` is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before `CONSTRAINT_ERROR` is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

### Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)

In assigning record types with discriminants, the expression appears to be evaluated in its entirety before `CONSTRAINT_ERROR` is raised when checking whether the expression's subtype is

compatible with the target's subtype. (See test C52013A.)

#### Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)

All choices are evaluated before CONSTRAINT\_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

#### Representation clauses.

An implementation might legitimately place restrictions on representation clauses used by some of the tests. If a representation clause is used by a test in a way that violates a restriction, then the implementation must reject it.

Enumeration representation clauses containing noncontiguous values for enumeration types other than character and boolean types are not supported. (See tests C35502I..J, C35502M..N, and A39005F.)

Enumeration representation clauses containing noncontiguous values for character types are not supported. (See tests C35507I..J, C35507M..N, and C55B16A.)

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are not supported. (See tests C35508I..J and C35508M..N.)

Length clauses with SIZE specifications for enumeration types are not supported. (See test A39005B.)

Length clauses with STORAGE\_SIZE specifications for access types are not supported. (See tests A39005C and C87B62B.)

Length clauses with STORAGE\_SIZE specifications for task types are not supported. (See tests A39005D and C87B62D.)

Length clauses with SMALL specifications are not supported. (See tests A39005E and C87B62C.)

Record representation clauses are not supported. (See test A39005G.)

## CONFIGURATION INFORMATION

Length clauses with SIZE specifications for derived integer types are not supported. (See test C87B62A.)

### Pragmas.

The pragma `INLINE` is not supported for procedures. The pragma `INLINE` is not supported for functions. (See tests LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F.)

### Input/output.

The package `SEQUENTIAL_IO` cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

The package `DIRECT_IO` cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

Modes `IN_FILE` and `OUT_FILE` are supported for `SEQUENTIAL_IO`. (See tests CE2102D and CE2102E.)

Modes `IN_FILE`, `OUT_FILE`, and `INOUT_FILE` are supported for `DIRECT_IO`. (See tests CE2102F, CE2102I, and CE2102J.)

`RESET` and `DELETE` are supported for `SEQUENTIAL_IO` and `DIRECT_IO`. (See tests CE2102G and CE2102K.)

Dynamic creation and deletion of files are supported for `SEQUENTIAL_IO` and `DIRECT_IO`. (See tests CE2106A and CE2106B.)

Overwriting to a sequential file truncates the file to last element written. (See test CE2208B.)

An existing text file can be opened in `OUT_FILE` mode, can be created in `OUT_FILE` mode, and can be created in `IN_FILE` mode. (See test EE3102C.)

More than one internal file can be associated with each external file for text I/O for reading only. (See tests CE3111A..E (5 tests), CE3114B, and CE3115A.)

More than one internal file can be associated with each external file for sequential I/O for reading only. (See tests CE2107A..D (4 tests), CE2110B, and CE2111D.)

More than one internal file can be associated with each external file for direct I/O for reading only. (See tests CE2107F..I (5 tests), CE2110E, and CE2111H.)



## CONFIGURATION INFORMATION

An internal sequential access file and an internal direct access file cannot be associated with a single external file for writing. (See test CE2107E.)

An external file associated with more than one internal file cannot be deleted for SEQUENTIAL\_IO, DIRECT\_IO, and TEXT\_IO. (See test CE2110B.)

Temporary sequential files are given names. Temporary direct files are given names. Temporary files given names are not deleted when they are closed. (See tests CE2108A and CE2108C.)

### Generics.

Generic subprogram declarations and bodies can be compiled in separate compilations. (See tests CA1012A and CA2009F.)

Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

## CHAPTER 3

### TEST INFORMATION

#### 3.1 TEST RESULTS

Version 1.9 of the ACVC comprises 3122 tests. When this compiler was tested, 24 tests had been withdrawn because of test errors. The AVF determined that 280 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for 10 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

#### 3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT -----	TEST CLASS						TOTAL -----
	A	B	C	D	E	L	
Passed	102	1048	1595	17	12	44	2818
Inapplicable	8	3	261	0	6	2	280
Withdrawn	3	2	18	0	1	0	24
TOTAL	113	1053	1874	17	19	46	3122

## TEST INFORMATION

### 3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER														TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14		
Passed	190	482	537	244	166	98	137	327	131	36	234	3	233	2818	
Inapplicable	14	91	138	4	0	0	6	0	6	0	0	0	21	280	
Withdrawn	2	13	2	0	0	1	2	0	0	0	2	1	1	24	
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122	

### 3.4 WITHDRAWN TESTS

The following 24 tests were withdrawn from ACVC Version 1.9 at the time of this validation:

B28003A	E28005C	C34004A	C35502P	A35902C
C35904A	C35A03E	C35A03R	C37213H	C37213J
C37215C	C37215E	C37215G	C37215H	C38102C
C41402A	C45614C	A74106C	C85018B	C87B04B
CC1311B	BC3105A	AD1A01A	CE2401H	

See Appendix D for the reason that each of these tests was withdrawn.

### 3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 280 tests were inapplicable for the reasons indicated:

- . C35502I..J (2 tests), C35502M..N (2 tests), C35507I..J (2 tests), C35507M..N (2 tests), C35508I..J (2 tests), C35508M..N (2 tests), A39005F, and C55B16A use enumeration representation clauses which are not supported by this compiler.
- . C35702A uses SHORT\_FLOAT which is not supported by this implementation.

- . C35702B uses LONG\_FLOAT which is not supported by this implementation.
- . A39005B and C87B62A use length clauses with SIZE specifications for derived integer types or for enumeration types which are not supported by this compiler.
- . A39005C..D (2 tests), C87B62B and C87B62D use length clauses with STORAGE\_SIZE specifications for access types or for task types which are not supported by this implementation.
- . A39005E and C87B62C use length clauses with SMALL specifications which are not supported by this implementation.
- . A39005G uses a record representation clause which is not supported by this compiler.
- . The following tests use LONG\_INTEGER, which is not supported by this compiler:

C45231C	C45304C	C45502C	C45503C	C45504C
C45504F	C45611C	C45613C	C45631C	C45632C
B52004D	C55B07A	B55B09C		

- . C45231D requires a macro substitution for any predefined numeric type other than INTEGER, SHORT\_INTEGER, LONG\_INTEGER, FLOAT, SHORT\_FLOAT, and LONG\_FLOAT. This compiler does not support any such types.
- . C45304A, C45504B and C45632B are ruled inapplicable for this implementation as subtypes of numeric types are treated with the same precision as the base type, hence intermediate results of the arithmetic operations in these tests lie within the range of the base type and so no exceptions are raised. This is in accordance with AVO regulations. (See LRM 11.6(6))
- . C45531M, C45531N, C45532M, and C45532N use fine 48-bit fixed-point base types which are not supported by this compiler.
- . C45531O, C45531P, C45532O, and C45532P use coarse 48-bit fixed-point base types which are not supported by this compiler.
- . B86001D requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.
- . C86001F redefines package SYSTEM, but TEXT\_IO is made obsolete by this new definition in this implementation and the test cannot be executed since the package REPORT is dependent on the package TEXT\_IO.

## TEST INFORMATION

- CA3004E, EA3004C, and LA3004A use the `INLINE` pragma for procedures, which is not supported by this compiler.
- CA3004F, EA3004D, and LA3004B use the `INLINE` pragma for functions, which is not supported by this compiler.
- AE2101C, EE2201D, and EE2201E use instantiations of package `SEQUENTIAL_IO` with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.
- AE2101H, EE2401D, and EE2401G use instantiations of package `DIRECT_IO` with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.
- CE2107B..E (4 tests), CE2107G..I (3 tests), CE2110B, CE2111D, CE2111H, CE3111B..E (4 tests), and CE3114B are inapplicable because multiple internal files cannot be associated with the same external file. The proper exception is raised when multiple access is attempted.
- The following 201 tests require a floating-point accuracy that exceeds the maximum of 15 digits supported by this implementation:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

### 3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required for 8 Class B tests, and 2 Class C tests.

The following Class B tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B22003A	B24007A	B24009A	B35302A	B38009B
B67001C	B95032A	B97103E		

The following Class C test evaluations were modified for the reasons indicated below:

- . C45304B is ruled passed as subtypes of numeric types are treated with the same precision as the base type by this implementation, hence intermediate results of the arithmetic operations in this test lie within the range of the base type and no exceptions are raised. This behaviour is in accordance with AVO regulations. (See LRM 11.6(6))
- . C4A012B is ruled passed as it raises `NUMERIC_ERROR` which is now accepted behaviour by the AVO.

### 3.7 ADDITIONAL TESTING INFORMATION

#### 3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the Siemens BS2000 Ada Compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

#### 3.7.2 Test Method

Testing of the Siemens BS2000 Ada Compiler using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a Siemens 7.570P operating under BS2000, V7.6 .

A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were included in their modified form on the magnetic tape.

The contents of the magnetic tape were loaded directly onto the host computer. After the test files were loaded to disk, the full set of tests was compiled and linked on the Siemens 7.570P, and all executable tests were linked and run.

## TEST INFORMATION

The compiler was tested using command scripts provided by Siemens AG, Muenchen and reviewed by the validation team. The compiler was tested using all default option settings except for the following:

Option	Effect
566	Suppress listing by Ada linker of all linked Ada compilation units.

Tests were compiled, linked, and executed (as appropriate) using a single host computer using three batch queues in parallel. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

### 3.7.3 Test Site

Testing was conducted at SIEMENS AG at Muenchen 83, Germany and was completed on 5th March 1988.

APPENDIX A  
DECLARATION OF CONFORMANCE

Siemens AG, Muenchen has submitted the following  
Declaration of Conformance concerning the Siemens  
BS2000 Ada Compiler.



## DECLARATION OF CONFORMANCE

Compiler Implementor: Siemens AG, Muenchen  
Ada Validation Facility: IABG m.b.H., Dept S71  
Ada Compiler Validation Capability (ACVC) Version: 1.9

### Base Configuration

Base Compiler Name and Version: Siemens BS2000 Ada Compiler V1.0  
Host Architecture ISA And OS&VER #: Siemens 7.570P BS2000/V7.6  
Target Architecture ISA And OS&VER #: Siemens 7.570P BS2000/V7.6

### Derived Compiler Registration

Derived Compiler Name And Version: Same as above  
Host Architecture ISA: 7.531, 7.536, 7.541, 7.551,  
7.530, 7.550, 7.560,  
7.561, 7.571, 7.550,  
7.560, 7.570, 7.580, 7.590,  
7.700  
OS&VER #: BS2000/V7.5, V7.6, V8.0, V8.5, V9.0, V9.2  
Target Architecture ISA And OS&VER #: Same as Host

### Implementor's Declaration

I, the undersigned, representing Siemens AG, Muenchen, have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler listed in this declaration. I declare that Siemens AG, Muenchen is the owner of record of the Ada language compiler listed above and, as such, is responsible for maintaining said compiler in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler listed in this declaration shall be made only in the owner's corporate name.

  
-----  
Siemens AG, Muenchen

Date: 5.4.88

-----  
Ada is a registered trademark of the United States Government  
(Ada Joint Program Office).

### Owner's Declaration

I, the undersigned, representing Siemens AG, Muenchen, take full responsibility for implementation and maintenance of the Ada compiler listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that all of the Ada language compilers listed, and their host/target performance, are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

Hoyer  
Siemens AG, Muenchen

Date: 5. 4. 88

## APPENDIX B

### APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the Siemens B52000 Ada Compiler, V1.0, are described in the following sections, which discuss topics in Appendix F of the Ada Standard. Implementation-specific portions of the package STANDARD are also included in this appendix.

package STANDARD is

...

type INTEGER is range -2\_147\_483\_647 .. 2\_147\_483\_647;

type SHORT\_INTEGER is range -32\_768 .. 32\_767;

type FLOAT is digits 15 range -2#1.0#E212 .. 2#1.0#E212;

type DURATION is delta 2#1.0#E-14 range -131\_071.0 .. 131\_071.0;

-- DURATION'SMALL = 2#1.0#E-14;

...

end STANDARD;

## F. Implementation-Dependent Characteristics

- 1 The *Ada language definition* allows for certain machine-dependences in a controlled manner. No machine-dependent syntax or semantic extensions or restrictions are allowed. The only allowed implementation-dependences correspond to implementation-dependent pragmas and attributes, certain machine-dependent conventions as mentioned in chapter 13, and certain allowed restrictions on representation clauses.
- 2 This appendix summarizes all implementation-dependent characteristics of the Siemens BS2000 Ada Compiler. It describes:
  - 3 (1) The form, allowed places, and effect of every implementation-dependent pragma.
  - (2) The name and the type of every implementation-dependent attribute.
  - (3) The specification of the package SYSTEM (see 13.7).
  - (4) The list of all restrictions on representation clauses (see 13.1).
  - (5) The conventions used for any implementation-generated name denoting implementation-dependent components (see 13.4).
  - (6) The interpretation of expressions that appear in address clauses, including those for interrupts (see 13.5).
  - (7) Any restriction on unchecked conversions (see 13.10.2).
  - (8) Any implementation-dependent characteristics of the input-output packages (see 14).

### F.1 Implementation-Dependent Pragmas

- 1 There are no implementation-defined pragmas.
- 2 The only language name accepted by pragma INTERFACE is ASSEMBLER.
- 3 The only priority accepted by pragma PRIORITY is represented by an expression of the static value 0 (cf. the definition of the subtype PRIORITY in package SYSTEM).

### F.2 Implementation-Dependent Attributes

- 1 There are no implementation-dependent attributes.

### F.3 Specification of the Package SYSTEM

package SYSTEM is

```

    type ADDRESS is new INTEGER;
    type NAME     is (BS2000);
    SYSTEM_NAME   : constant NAME := BS2000;
    STORAGE_UNIT  : constant := 8;
    MEMORY_SIZE   : constant := 4_000_000;

    -- System-Dependent Named Numbers:
    MIN_INT       : constant := -2_147_483_647;
    MAX_INT       : constant := 2_147_483_647;
    MAX_DIGITS    : constant := 15;
    MAX_MANTISSA  : constant := 31;
    FINE_DELTA    : constant := 2.0**(-31);
    TICK         : constant := 0.000_1;

    -- Other System-Dependent Declarations
    subtype PRIORITY is INTEGER range 0 .. 0;

```

end SYSTEM;

### F.4 Restrictions on Representation Clauses

Representation clauses are not yet supported.

### F.5 Conventions for Implementation-Generated Names Denoting Implementation-Dependent Components in Record Representation Clauses

Implementation-dependent components may be added to record objects by the compiler. These components remain inaccessible to the user, i.e. they cannot be accessed via implementation-generated names, in particular they cannot be referred to in record representation clauses.

### F.6 Interpretation of Expressions Appearing in Address Clauses

Address clauses are not yet supported.

## F.7 Restrictions on Unchecked Type Conversions

- 1 The Siemens BS2000 Ada compiler supports the generic function UNCHECKED\_CONVERSION with the following restriction:
- 2 The actual generic subtype corresponding to the formal generic type TARGET must not be an unconstrained array type, and it must not be an unconstrained type with discriminants that have no defaults. The instances gained from UNCHECKED\_CONVERSION return a target value whose bit pattern is a left-aligned copy of that of the source value. The number of bits transferred corresponds to the size of the target subtype. If the size of the source value is greater than the size of the target subtype, then the source value information is truncated on the right hand side, i.e. the low order bits are ignored. If the size of the source value is not greater than the size of the target subtype, then - again - as many bits are transferred as corresponds to the size of the target subtype, and no padding with zeroes, spaces or other characters is performed.

## F.8 Implementation-Dependent Characteristics of the Input-Output Packages

### F.8.1 Introduction

- 1 The SEQUENTIAL\_IO, DIRECT\_IO and TEXT\_IO packages are all written in Ada and they make calls on BASIC\_IO which is a "typeless" package working with addresses and byte counts. SEQUENTIAL\_IO and DIRECT\_IO are generic packages, also INTEGER\_IO, FIXED\_IO, FLOAT\_IO and ENUMERATION\_IO in TEXT\_IO are generic.
- 2 The routines written in assembler language have the name ADARTSBx with x in 1 .. 9, A .. Q, while the BASIC\_IO routines have nearly the same names as in the input-output packages of Ada.

### F.8.2 Conventions for NAME and FORM

- 1 External files are supported by the SAM, ISAM, SYSDTA, SYSOUT and SYSLST BS2000 files where the value of the parameter FORM of the CREATE and OPEN procedures determines which access method is selected.
- 2 The set of allowable values of FORM is given below together with the type of BS2000 file corresponding to it. Leading blanks and lower-case letters are not allowed in the FORM string.
- 3 value of FORM BS2000 access method

SAM	Sequential Access Method
ISAM	Indexed Sequential Access Method
SYSDTA	The file (or device) associated with the BS2000 system file SYSDTA
SYSOUT	The file (or terminal) associated with the BS2000 system file SYSOUT
SYSLST	The file (or printer) associated with the BS2000 system file SYSLST
SAM_PRINT	Like SAM but with printer control characters in the first column (see below)
ISAM_PRINT	Like ISAM but with printer control characters in the first column (see below)

## Implementation-Dependent Characteristics

- 4 Each input-output package operates on a subset of the allowable forms.
- 5 SAM, ISAM, SAM\_PRINT and ISAM\_PRINT files are identified by the value of the parameter NAME of the CREATE and OPEN procedures whose characters must conform to the BS2000 file naming conventions as described below. The value of the parameter NAME is ignored for other values of FORM.
- 6 The syntax associated with the string NAME is as follows

```
NAME          ::= .link_name| file_name
file_name     ::= :cat_id: $ user_id . name { . name } |
                $user_id . name { . name } |
                $admin_name |
                name { . name }

cat_id        ::= name_character
link_name     ::= name_character { name_character }
user_id       ::= name_character { name_character }
admin_name    ::= name_character { name_character }
name          ::= name_character { name_character }

name_character ::= upper_case_letter | digit |
                  special_character

special_character ::= $ | @ | # | -
```

- 7 BS2000 imposes the following additional restrictions upon the syntax of NAME.
  1. The maximum length of a link\_name or a user\_id is eight characters.
  2. The maximum length of a file\_name starting with :cat\_id: is 54 characters.
  3. The maximum length of a file\_name starting with \$user\_id is 51 characters.
  4. The maximum length of an admin\_name is 47 characters unless it contains one or more periods in which case the maximum length is 53 characters.
  5. The maximum length of a file\_name starting with name is 41 characters.
  6. The first character of a name must not be a special character, and the last character must not be a hyphen.
  7. A file\_name must include at least one letter.

- 8 *Example of using TEXT\_IO:*

```
with TEXT_IO; use TEXT_IO;
package FILE_MANAGEMENT is

    ACTUAL_FILE1 : TEXT_IO.FILE_TYPE;
    ACTUAL_FILE2 : TEXT_IO.FILE_TYPE;
    ACTUAL_FILE3 : TEXT_IO.FILE_TYPE;

begin
    -- Create a BS2000-SAM file with name A.SAM.FILE
```

```

TEXT_IO.CREATE      (FILE   => ACTUAL_FILE1,
                    MODE   => OUT_FILE,
                    NAME   => "A.SAM.FILE",
                    FORM   => "SAM");

--      Create a BS2000-ISAM file with the link name ABCD and with file_name
--      AN.ISAM.FILE
--      BS2000 command: "/FILE AN.ISAM.FILE, LINK = ABCD" (Note: no '.').

TEXT_IO.CREATE      (FILE   => ACTUAL_FILE2,
                    MODE   => OUT_FILE,
                    NAME   => ".ABCD",      -- Note: with '.'
                    FORM   => "ISAM");

--      Open the BS2000-SAM file with link_name XYZ and with file_name A.SAM.FILE
--      BS2000 command: "/FILE A.SAM.FILE, LINK = XYZ" (Note: no '.').

TEXT_IO.OPEN        (FILE   => ACTUAL_FILE3,
                    MODE   => IN_FILE,
                    NAME   => ".XYZ",      -- Note: with '.'
                    FORM   => "SAM");

end FILE_MANAGEMENT;

```

### F.8.3 File management

- 1 This section describes the implementation restrictions which apply to the sequential, direct and text input-output packages equally. The maximum number of objects which may be stored in an external file is dependent upon the maximum number of records or the maximum number of blocks which may be stored in its underlying BS2000 file. The values given below state this maximum for each FORM provided that limits imposed by the system configuration are not otherwise reached. For the direct and sequential input-output packages, each object is stored in a separate record or block; for the text input-output package, each line is stored in a separate record.
- 2

FORM	Maximum Number of Records / Blocks
SAM	configuration dependent limit
ISAM	99 999 999 records
SAM_PRINT	configuration dependent limit
ISAM_PRINT	99 999 999 records
SYSDTA	configuration dependent limit
SYSOUT	configuration dependent limit
SYSLSL	configuration dependent limit
- 3 Two alternative record formats are available for ISAM and SAM files, varying and constant length. TEXT\_IO always uses varying length records whereas DIRECT\_IO and SEQUENTIAL\_IO support both formats. A varying length record format is used if an instance of direct or sequential input-output packages uses unconstrained element-types. Otherwise a fixed length record format is used where the length equals the value of  $(\text{ELEMENT TYPE'SIZE} + 7) / 8$  (that are the number of bytes needed for this special type).



## Implementation-Dependent Characteristics

- 4 The maximum size of the objects which can be stored in an external file is restricted. The universal integer value which results from the application of the SIZE attribute to every object accessed by the package must lie within a range which is dependent upon the FORM and whether constant or varying size records are being used. The exception USE\_ERROR is raised if this constraint is violated.

5	FORM	constant/varying	OBJECT'SIZE (bits)
	SAM	constant	1 .. 16 384
	SAM	varying	1 .. 16 352
	SAM_PRINT	varying	1 .. 16 352
	ISAM	constant	1 .. 16 320
	ISAM	varying	1 .. 16 288
	ISAM_PRINT	varying	1 .. 16 288
	SYSDTA	varying	1 .. 2 032
	SYSOUT	varying	1 .. 2 032
	SYSLST	varying	1 .. 2 032

- 6 The default value in TEXT\_IO for the FORM parameter is SAM\_PRINT, in SEQUENTIAL\_IO it is SAM, while in DIRECT\_IO it is ISAM.

- 7 SAM and ISAM files with no null string for NAME are permanent files in that their lifetimes are independent of the currently running Ada program and of the BS2000 tasks in which they were created. Permanent files may be closed in one BS2000 task and opened subsequently in the same or another task without loss of their contents (for MODE = IN or IN\_OUT).

- 8 A null string for NAME specifies an external file that is not accessible after the completion of the main program (a temporary file).

- 9 The BS2000 names for temporary files are

"#TEMP.xxxx.yymmdd.zzzzzz.nnnnnnn" with

xxxx	::= decimal number (tsn of the current BS2000 task)
yymmdd	::= decimal number (current date)
zzzzzz	::= decimal number (time in seconds filled up with leading 0)
nnnnnnn	::= decimal number (range 1000000 .. 9999999).

- 10 When a SAM or ISAM file (selected by the value of NAME) is opened, there is no check that the form of the BS2000 file corresponds to the value of the FORM parameter of the OPEN procedure. There is no check either that the input-output package opening a SAM or ISAM file is the same package as the one which created the file. If either of these conditions is violated, the program may deliver unexpected results.

- 11 SYSDTA, SYSOUT and SYSLST files are temporary files whose lifetime ends with that of the BS2000 task which created them.

- 12 The SYSDTA, SYSOUT and SYSLST files are unique within a BS2000 task. SYSDTA and SYSOUT are opened by the elaboration of TEXT\_IO. SYSDTA is the FORM of the Ada STANDARD\_INPUT file, while SYSOUT is the FORM of the Ada STANDARD\_OUTPUT file. The user may open SYSDTA at most once at a time additionally to STANDARD\_INPUT. Also only one file may be opened at a time with FORM parameter SYSLST. Opening a file with these FORM parameters causes a SYSFILE command for the BS2000 system. Therefore, reading from the BS2000 system file SYSDTA is equivalent to reading from STANDARD\_INPUT, but both have their own FCB. A close on a file with the FORM parameter SYSDTA or SYSLST causes a redirection of the BS2000 system files SYSDTA and SYSLST to (PRIMARY)

via a SYSFILE command. The user may redirect SYSDTA or SYSLST to (PRIMARY), too, by opening a file with the FORM parameter SYSDTA or SYSLST and NAME parameter (PRIMARY). A SYSOUT file may not be opened or deleted because its redirection is impossible and STANDARD\_OUTPUT is opened with the FORM parameter SYSOUT during the elaboration.

- 13 No assumptions should be made about the way objects are stored in the various BS2000 files except as described for the TEXT\_IO package. For example, the mapping of indices onto ISAM keys may differ between different versions of the input-output packages.

### F.8.3.1 SEQUENTIAL\_IO

- 1 The value of the FORM parameter of the CREATE and OPEN procedures is restricted to SAM.
- 2 The package SEQUENTIAL\_IO cannot be instantiated with unconstrained array types and unconstrained record types with discriminants that have no defaults.

### F.8.3.2 DIRECT\_IO

- 1 The value of the FORM parameter of the CREATE and OPEN procedures is restricted to ISAM.
- 2 The package DIRECT\_IO cannot be instantiated with unconstrained array types and unconstrained record types with discriminants that have no defaults.
- 3 The value of an index may be set in the range 1 .. INTEGER'LAST.

### F.8.3.3 TEXT\_IO

- 1 The value of the FORM parameter of the CREATE procedure is restricted to SAM, ISAM, SAM\_PRINT and ISAM\_PRINT, while the OPEN procedure may use SYSLST and SYSDTA additionally.
- 2 The lines contained in text files are variable in length in the range 1 .. 2000 characters. The upper bound for the subtype FIELD is 500;
- 3 The upper bound for the type COUNT is INTEGER'LAST.
- 4 In printable files (FORM = SAM\_PRINT, FORM = ISAM\_PRINT) lines are stored in the second to 2001st character of a BS2000 variable length file record. The ASCII characters of the Ada program are represented by their corresponding EBCDIC characters in the BS2000 files. The first character of the record is a printer control character where ' ' means line-feed and 'A' page feed. Thus BS2000 files created by a call to TEXT\_IO can be printed using the /PRINT command (with SPACE = E) or displayed using the EDOR and EDT text file editors. The printer control characters are used to implement the line and page terminators and can be manipulated using the standard line and page control procedures. The transfer of lines to BASIC\_IO is done by NEW\_LINE, NEW\_PAGE, CLOSE and RESET - or if a line is filled up.

## Implementation-Dependent Characteristics

- 5 An empty line after a page terminator is identified by an EBCDIC.NUL in the second column. Other empty lines are identified by an EBCDIC.SOH in the first and a EBCDIC.BLANK in the second column. Since TEXT\_IO converts ASCII.NUL to EBCDIC.NUL this special character may not be used in the first column of the first line of a new page as the only character in this line (that is the second column of the BS2000 file).
- 6 Two FORM parameters (SAM and ISAM) may be used by TEXT\_IO to support files without printer control characters in the first column.
- 7 In these files the end of a line is interpreted as a line terminator. A page terminator is an EBCDIC.NUL in the last column of the line. Therefore the user may not output an EBCDIC.NUL to the last column of a line without an incrementation of the current page by TEXT\_IO on reading the file again. An EBCDIC.STX in the first column designates an empty line.
- 8 The MODE of the SYSLST file is restricted to OUT\_FILE.  
The MODE of the SYSDTA file is restricted to IN\_FILE.
- 9 The standard input file has the FORM SYSDTA and the standard output file has the FORM SYSOUT. In the dialogue mode of BS2000 a call of NEW\_PAGE on STANDARD\_OUTPUT causes the deletion of the screen and a call of NEW\_LINE with an empty internal buffer causes the output of a line feed.
- 10 The transfer of characters from TEXT\_IO to BASIC\_IO is done line by line. All characters are stored in an internal buffer. The line is displayed after calling PUT\_LINE, NEW\_LINE, NEW\_PAGE, CLOSE and RESET. On the other hand the terminal represents the two distinct files STANDARD\_OUTPUT and STANDARD\_INPUT in one "file" (terminal). Therefore, a sequence of PUT - GET - PUT routine calls without calling NEW\_LINE or NEW\_PAGE causes the following display sequence at the terminal.

11 *Example:*

```
with TEXT_IO; use TEXT_IO;
package DIALOGUE is
begin
    ...

    PUT (STANDARD_OUTPUT," THE USER AT THE TERMINAL IS ");
    GET (STANDARD_INPUT, USER_NAME);    -- the user enters "TOM WHO IS NOT"
    PUT (STANDARD_OUTPUT, " CRAZY");
    NEW LINE (STANDARD_OUTPUT);

end DIALOGUE;
```

12 *Example of the interaction (characters typed by the user are italicized):*

```
                -- the user has to enter something, assume he enters "TOM WHO IS NOT"
*TOM WHO IS NOT
THE USER AT THE TERMINAL IS CRAZY
```

13 The user intended to get:

```
THE USER AT THE TERMINAL IS
*TOM WHO IS NOT
CRAZY
```

## Implementation-Dependent Characteristics

- 14 The user should use in those cases a sequence of PUT\_LINE - GET - PUT - NEW LINE. Then the example would display:

THE USER AT THE TERMINAL IS  
\*TOM WHO IS NOT  
CRAZY

- 15 A text file read from a terminal (via SYSDTA) is handled like a file with FORM SAM, that means page terminators and empty lines are recognized by an EBCDIC.NUL or EBCDIC.STX as described above for SAM files.

# APPENDIX C

## TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .IST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

<u>Name_and_Meaning_____</u>	<u>Value_____</u>
\$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	(1..239 => 'A', 240 => '1')
\$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	(1..239 => 'A', 240 => '2')
\$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	(1..120=>'A',121=>'3',122..240=>'A')
\$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1..120=>'A',121=>'4',122..240=>'A')
\$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1..237=>'0',238..240=>"298")

# TEST PARAMETERS

Name and Meaning	Value
<b>\$BIG_REAL_LIT</b> A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	(1..235=>'0',236..240=>'690.0')
<b>\$BIG_STRING1</b> A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	'1=>'',2..121=>'A',122=>'')
<b>\$BIG_STRING2</b> A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	(1=>'',2..120=>'A',121=>'1',122=>'')
<b>\$BLANKS</b> A sequence of blanks twenty characters less than the size of the maximum line length.	(1..220 => ' ')
<b>\$COUNT_LAST</b> A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2_147_483_647
<b>\$FIELD_LAST</b> A universal integer literal whose value is TEXT_IO.FIELD'LAST.	240
<b>\$FILE_NAME_WITH_BAD_CHARS</b> An external file name that either contains invalid characters or is too long.	BAD_FILE_NAME
<b>\$FILE_NAME_WITH_WILD_CARD_CHAR</b> An external file name that either contains a wild card character or is too long.	WILD*FILE*NAME
<b>\$GREATER_THAN_DURATION</b> A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	131_071.5

## TEST PARAMETERS

Name and Meaning	Value
\$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	200_000.0
\$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	BAD_FILE_NAME
\$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	MUCH-TOO-LONG-NAME-FOR-A-CORRECT-BS2000-FILE
\$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-2_147_483_647
\$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	2_147_483_647
\$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	2_147_483_648
\$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-131_071.5
\$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-200_000.0
\$MAX_DIGITS Maximum digits supported for floating-point types.	15
\$MAX_IN_LEN Maximum input line length permitted by the implementation.	240
\$MAX_INT A universal integer literal whose value is SYSTEM.MAX_int.	2_147_483_647
\$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2_147_483_648

# TEST PARAMETERS

Name and Meaning	Value
<b>\$MAX_LEN_INT_BASED_LITERAL</b> A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1..2=>"2:",3..237=>'0',238..240=>"11:")
<b>\$MAX_LEN_REAL_BASED_LITERAL</b> A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1..3=>"16:",4..236=>'0',237..240=>"F.E:")
<b>\$MAX_STRING_LITERAL</b> A string literal of size MAX_IN_LEN, including the quote characters.	(1=>'"',2..239=>'A',240 =>'")')
<b>\$MIN_INT</b> A universal integer literal whose value is SYSTEM.MIN_INT.	-2_147_483_647
<b>\$NAME</b> A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	\$NAME
<b>\$NEG_BASED_INT</b> A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	8#37_777_777_776#



APPENDIX D  
WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 24 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

- B28003A: A basic declaration (line 36) wrongly follows a later declaration.
- E28005C: This test requires that 'PRAGMA LIST (ON);' not appear in a listing that has been suspended by a previous "pragma LIST (OFF);"; the Ada Standard is not clear on this point, and the matter will be reviewed by the ARG.
- C34004A: The expression in line 168 wrongly yields a value outside of the range of the target type T, raising CONSTRAINT\_ERROR.
- C35502P: Equality operators in lines 62 and 69 should be inequality operators.
- A35902C: Line 17's assignment of the nominal upper bound of a fixed-point type to an object of that type raises CONSTRAINT\_ERROR, for that value lies outside of the actual range of the type.
- C35904A: The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT\_ERROR, because its upper bound exceeds that of the type.

## WITHDRAWN TESTS

- C35A03E, These tests assume that attribute 'MANTISSA returns C  
and R: when applied to a fixed-point type with a null range,  
but the Ada Standard doesn't support this assumption.
- C37213H: The subtype declaration of SCONS in line 100 is wrongly  
expected to raise an exception when elaborated.
- C37213J: The aggregate in line 451 wrongly raises  
CONSTRAINT\_ERROR.
- C37215C, Various discriminant constraints are wrongly expected to  
E, G, H: be incompatible with type CONS.
- C38102C: The fixed-point conversion on line 23 wrongly raises  
C41402A: CONSTRAINT\_ERROR. 'STORAGE\_SIZE is wrongly applied to an  
object of an access type.
- C45614C: REPORT.IDENT\_INT has an argument of the wrong type  
(LONG\_INTEGER).
- A74106C, A bound specified in a fixed-point subtype declaration  
C85018B, lies outside of that calculated for the base type,  
C87B04B, raising CONSTRAINT\_ERROR. Errors of this sort occur at  
CC1311B: lines 37 and 59, 142 and 143, 16 and 48, and 252 and  
253 of the four tests, respectively (and possibly  
elsewhere).
- BC3105A: Lines 159..168 are wrongly expected to be illegal; they  
are legal.
- AD1A01A: The declaration of subtype INT3 raises CONSTRAINT\_ERROR  
for implementations that select INT'SIZE to be 16 or  
greater.
- CE2401H: The record aggregates in lines 105 and 117 contain the  
wrong values.